International Journal of LNCT, Vol 2(7)
ISSN (Online): 2456-9895

Cache memory management in big data

Rajesh Kumar [1] and Manish Shrivastava[2]
M.Tech Scholar, LNCT, Bhopal
Director of research and Development, LNCT, Bhoapl

Abstract
Big data processing consumes resources at very large scale and its very challenging to manage the memory for each running process. Researchers have developed the some memory management schemes which can synchronize the data flow for each process. Processes use two memories i.e. Main memory which retains only current processing data and cache which retains frequent required data for processes. It is very challenging to replace the cache data with new one because, during the processing of large volume data, process may claim any data block, so due to random block replacement, process may have to wait for a long time which may result unnecessarily delay. In this paper, we will explore the current research work related to cache management.

Keywords
Big Data, Cache, Memory, Resource management.

1.Introduction
THE explosion of Big Data has prompted much research to develop systems to support ultra-low latency service and real-time data analytics. Existing disk-based systems can no longer offer timely response due to the high access latency to hard disks. Big data can be referred as huge collection of data. Its main resources are Public/Govt. data, internet, social media, news channels, educational institutes, industries etc. Data is stored on the disks and for analysis purpose; it is loaded in to memory which has a limited size. So traditional system's memory is not suitable for big data processing in real time environment. Traditional systems have different types of memory i.e. volatile and non volatile memory and volatile memory consists of main memory and cache. Cache refers to the intermediate data that is produced by worker nodes/processes during the execution of a Map Reduce task. Redundancy techniques such as RAID[5] have been widely used in hard disk-based storage systems to offer fault-tolerance. RAID-1 is basically the same as data replication, which achieves very good reliability but requires a double cost. RAID-5 and RAID-6 improve the storage efficiency at the expense of decreasing access performance when faced with disk failures. But currently, how these redundancy techniques work in memory is still unknown and worth studying. "Big Data" is a hot research topic apart from the "Cloud" these years. Since the growth rate of digital data has been faster and faster, coping with data storage, query, and analysis derived from big data has become an important research issue. In addition, how these huge amount and variable types of data are processed together with an appropriate data mining technique to derive useful information is the reason why big data has been paid attention to worldwide. In the past, only a few applications required massive data processing, such as weather forecast, genetic codes, and financial transactions. However, more and more industries are facing the difficulties of processing and analyzing big data due to the great progress of technology. For instance, there are more than one billion times of product search on eBay every day, and, at the same time, the total capacity of the databases has already reached 10 PB with 1.5 trillion data growth per day. Because the new information is generating every day and the size of database is very large, it's difficult to analyze the consumer behavior and browsing habits of their customers. Likewise, hundreds of millions of people use Facebook every day leads to generating a huge amount of data in the databases, most of them being images and videos that are rarely processed in the traditional databases. The people who are engaged in social network business are facing the challenges of how to manage the databases and cope with those businesses to understand the consumer trends. One of the successful cases is Walmart, the biggest chain store in the United States. According to the analysis of customers' bill data, Walmart makes the decision to put beer and diapers together, results in the increase of beer sales. Therefore, the data processing and applications of big data have become a very important issue of businesses. Many companies develop two approaches to resolve the problem of manipulating big data in databases. The first approach is to abandon the relational database and employed the NoSQL database. NoSQL database is a generic term and refers to non-relational database technology including several different types of database systems. Most of these database systems do not support the standard SQL query, for example, the Google's BigTable and Microsoft's Azure cloud platform. However, it requires a lot of manpower and costs to migrate the SQL database to NoSQL architecture. Thus, the second approach is to improve

performance of relational database and provide a flexible expansion. Since database is accessed mainly through a number of queries, it is critical to shorten the time required for database query. This paper provides a hybrid cache approach aiming at improving the query efficiency of the relational database system. We conduct the performance improvements by adopting UUID (Universally Unique Identifier) indexing values and hierarchical caching methodology. The scheme proposed allows us to accelerate the database queries without affecting the structures of the original SQL programming framework. Experimental results demonstrate that our proposed method can significantly speed up the database query, improving the performance of a big database.

2.Background and related work

Technologies for in-memory systems : This section, we shall introduce some concepts and techniques that are important for efficient in-memory data management, including memory hierarchy, non-uniform memory access (NUMA), transactional memory, and non-volatile random access memory (NVRAM). These are the basics on which the performance of in memory data management systems heavily rely. Hierarchy The memory hierarchy is defined in terms of access latency and the logical distance to the CPU. In general, it consists of registers, caches (typically containing L1 cache, L2 cache and L3 cache), main memory (i.e., RAM) and disks.When a new object needs to be stored and the cache is full, a replacement algorithm must be used to determine which object should be removed to make space for the new objects. Cache replacement algorithms decide which data could stay in the cache. An outstanding cache replacement algorithm has few number of misses in the cache and can improve the cache hit ratio and byte ratio too. A lot of academic and industry researchers are studying toward finding the cache replacement policy. At present, cache replacement algorithms mainly focus on the following categories: Least Recently Used Algorithm, Least Frequently Used Algorithm, SIZE Algorithm, Function-based Algorithm, Randomized-based Algorithm and Weighting-based Algorithm.

(1) Least recently used algorithm:- In this type of algorithm, time is the most primary factor. Least Recently Used (LRU) algorithm is one of the most popular policies in web cache. Its main idea is that if the data has been requested recently, the probability of it being requested in the future will be higher. It is used to cache web browser information, like pictures, etc. When cache is full, it removes the least recently referenced objects. The advantage of this algorithm is that it is easy to implement and has a good performance in client-side. But it only considers the last referenced time while ignoring the frequency of references for a certain web object.

(2) Least frequently used (LFU) algorithm:- In this type of algorithm, researchers consider the object popularity (or frequency count) as the primary factor. Its main idea is that if the data has been requested more frequently, the probability of it being requested in the future will be higher. It usually caches the web URLs. When cache is full, it removes web object with the lowest frequency. Its advantage is that it is easy to implement too and has a good performance in proxy side. But if one object has high frequency previously, it will be stored in the cache although it will no longer be accessed. We call it ‘‘cache pollution’’.

(3) SIZE algorithm:- In this type of algorithm, object size is treated as the primary factor when caching the object. Its main idea is that if the data has a bigger size, it may occupy more cache space. When cache is full, it removes web object with the biggest size. This type of algorithm usually caches the entity objects those with real object size rather than urls, etc. When the size is the same, it usually considers time as the second factor. Its advantage is that it can remove the big size objects timely. But for those objects with small size in the cache, they will be stored in the cache although they would not be accessed. Then it will also cause ‘‘cache pollution’’.

(4) Function-based algorithm:- This type of cache replacement algorithm calculates the cache value of every web object [5]. For example, Greedy Dual-Size algorithm calculates the object size, access latency, cost and etc. When the cache is full, it removes the object according to the smallest cache value. The cache values is calculated as: $Ki = Ci/Si + L$ (1) Where Ci is the cost of retrieved object i, Si is the size of object i, L is the aging factor. However, the method ignores the frequency and other factors.

(5) Randomized-based algorithm:- In this type of algorithm, no factor is treated as the primary factor when caching the object. Its main idea is that every cache objects has the equal value. When cache is full, it selects a random object and removes. It is easy to implement, but it has low performance and its hit rate is unsatisfactory. Because server doesnt need analyze the objects attribute, it just remove an object randomly. But the objects are impossible to request an object only once, so it may remove the data that we call ‘‘hot data’’.

However, accessing data from registers is very much faster. Registers play the role as the storage containers that CPU uses to carry out instructions, while caches act as the bridge between the registers and main memory due to the high transmission delay between the registers and main memory. Cache is made of high-speed static RAM (SRAM) instead of slower and cheaper dynamic RAM (DRAM) that usually forms the main memory. In general, there are three levels of caches, i.e., L1 cache,L2 cache and L3 cache (also called last level cache—LLC), with increasing latency and capacity. L1 cache is further divided into data cache (i.e., L1-dcache) and instruction cache (i.e., L1-icache) to avoid any interference between data access and instruction access. place in the cache, which makes addressing faster. Under fully associative strategy, each entry can be put in any place, which offers fewer cache misses. The N-way associative strategy is a compromise between direct mapping and fully associative-it allows each entry in the memory to be in any of N places in the cache, which is called a cache set. N-way associative is often used in practice, and the mapping is deterministic in terms of cache sets.

Figure 1 Mapping is deterministic in terms

In addition, most architectures usually adopt a leastrecently-used (LRU) replacement strategy to evict a cache line when there is not enough room. Such a scheme essentially utilizes temporal locality for enhancing performance. the latency to access cache is much shorter than the latency to access main memory. In order to gain good CPU performance, we have to guarantee high cache hit rate so that high-latency memory accesses are reduced. In designing an in-memory management system, it is important to exploit the properties of spatial and temporal locality of caches. For examples, it would be faster to access memory sequentially than randomly, and it would also be better to keep a frequently-accessed object resident in the cache. The advantage of sequential memory access is reinforced by the prefetching strategies of modern CPUs. Main Memory and Disks Main memory is also called internal memory, which can be directly addressed and possibly accessed by the CPU, in contrast to external devices such as disks. Main memory is usually made of volatile DRAM, which incurs equivalent latency for random accesses without the effect of caches, but will lose data when power is turned off. Recently, DRAM becomes inexpensive and large enough to make an in-memory database viable. Even though memory becomes the new disk [1], the volatility of DRAM makes it a common case that disks3 are still needed to backup data. Data transmission between main memory and disks is conducted in units of pages, which makes use of data spatial locality on the one hand and minimizes the performance degradation caused by the highlatency of disk seek on the other hand. A page is usually a multiple of disk sectors4 which is the minimum transmission unit for hard disk. In modern architectures, OS usually keeps a buffer which is part of the main memory to make the communication between the memory and disk faster.The buffer is mainly used to bridge the performance gap between the CPU and the disk. It increases the disk I/O performance by buffering the writes to eliminate the costly disk seek time for every write operation, and buffering the reads for fast answer to subsequent reads to the same data. In a sense, the buffer is to the disk as the cache is to the memory. And it also exposes both spatial and temporal locality, which is an important factor in handling the disk I/O efficiently. Memory Hierarchy Utilization This section reviews related works from three perspective-register-conscious optimization.

Problem formulation: Big Data processing requests consume lot of resources over system. Large scale buffer is required for data processing because there are several input/output operations may exists those can be executed parallel. Frequent required data can be temporarily stored in a buffer, called cache but it is not feasible to store huge volume of random data n cache due to its cost although cache can reduce the number of input and output operations. For large scale data, it is still a major issue that how to optimize the cache in such a way that service consumer process should perform minimum buffer operations by making the use of cache memory. Common Cache management Issues: Cache configuration State Maintenance for the cached object.

3.Conclusion

In this survey paper, cache management issues were investigated. Researchers have developed various solutions to manage the cache for Big Data processing and each has its own merit sand demerits. Now we discuss most relevant solutions for cache management, i.e. Data Aware Caching, Dache is a framework based on Mapreduce MAP Reduce. It is able to identify the duplicate tasks for elimination purpose. HiBench workloads can be used to analyze the performance of Map reduce framework, HBase Tree based data model splits the data and data repositioning is used to process large volume data by dynamic loading and local cache management. It can be further enhanced to process the distributed geological data. K-means clustering and PageRank offers fast cache solution which works on the combination of Hadoop and PIG frameworks and it splits data into multiple tuples which are further processed by Mappers. Optimization of CPU based I/O operations can reduce the requirements of available bandwidth. Memory FastForward reduces the total memory traffic and optimizes energy consumption. Smart cache can reduce the total process execution time. This study can be further utilized to develop a solution for cache management.

References

[1] Zhao Y, Wu J, Liu C. Dache: A data aware caching for big-data applications using the MapReduce Framework. Tsinghua Science and Technology. 2014; 19(1):39-50.

[2] Tamboli S, Patel SS. A survey on innovative approach for improvement in efficiency of caching technique for big data application. In pervasive computing (ICPC), 2015 international conference on 2015 (pp. 1-6). IEEE.

[3] Ahmed ST, Loguinov D. Modeling randomized data streams in caching, data processing, and crawling applications. In computer communications (INFOCOM), 2015 IEEE Conference on 2015 (pp. 1625-33). IEEE.

[4] https://www.gridgain.com/. Accessed 26 July 2018.

[5] Neumeyer L, Robbins B, Nair A, Kesari A. S4: Distributed stream computing platform. InData Mining Workshops (ICDMW), 2010 IEEE International Conference on 2010 (pp. 170-7). IEEE.

[6] https://storm. incubator.apache.org/. Accessed 26 July 2018.

[7] Fitzpatrick B. Distributed caching with memcached. Linux Journal. 2004; 2004(124):5.

[8] Nishtala R, Fugal H, Grimm S, Kwiatkowski M, Lee H, et al. Scaling Memcache at Facebook. Innsdi 2013 (pp. 385-98).

[9] https://www.enterprisetech.com/2014/09/15/%20facebook-opens-tools-scale-memcached/ Accessed 26 July 2018.

[10] Tanenbaum AS, Van Steen M. Distributed systems: principles and paradigms. Prentice-Hall; 2007.